5-2021

Evolving Efficient Floor Plans For Hospital Emergency Rooms

Alex Ramsey
waramsey@unomaha.edu

Evolving Efficient Floor Plans For Hospital Emergency Rooms

By

Alex Ramsey

Undergraduate of Computer Science

University of Nebraska at Omaha

Abstract

Genetic Algorithms find wide use in optimization problems across many fields of research, including crowd simulation. This paper proposes that genetic algorithms could be used to create better floor plans for hospital emergency rooms, potentially saving critical time in high risk situations. The genetic algorithm implemented makes use of a hospital-specific crowd simulation to accurately evaluate the effectiveness of produced layouts. The results of combining genetic algorithms with a crowd simulation are promising. Future work may improve upon these results to produce better, more optimal hospital floor plans.

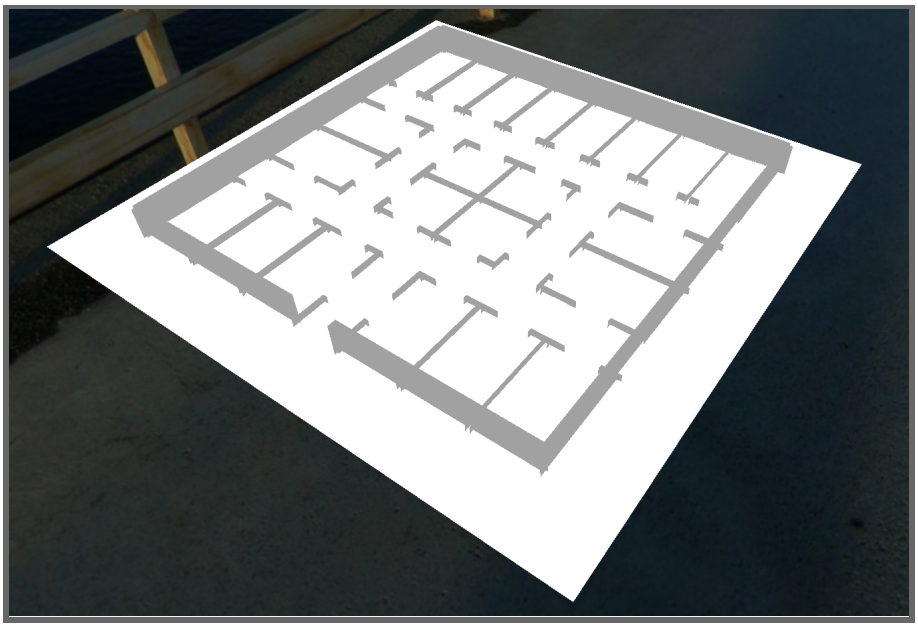


Fig 1. An emergency room floor plan produced by this research.

Table of Contents

Abstract 2

Table of Contents 3

Introduction 4

Background 6

Theory 8

Genetic Algorithms, Procedural Floor Plans, and Crowd Simulation 8

Heuristics and Fitness Functions 12

Implementation 14

Procedural Floor Plan Generation Algorithm 14

Genetic Algorithm 17

Results 24

Future Work and Conclusion 32

Acknowledgements 34

References 35

I. Introduction

In the field of crowd simulation, research efforts are directed towards producing accurate models of crowd movements through virtual environments. Though the individual purpose of these research efforts is distinct from other projects in crowd simulation, the research collectively works toward a greater goal. Generally speaking, crowd simulation is about understanding the behavior of individuals expressed as motion in a crowd as they interact with a given environment. Picking this goal apart, three major areas of research are apparent; the individual behaviors, "swarm behaviors," and environment are all common focuses of crowd simulation research.

The focus of this research paper is on the environment through which a crowd moves. A greater understanding of virtual environments in crowd simulations is important to our understanding of real world environments. It is my hope that, through the research presented in this paper, I may contribute to the collective knowledge and understanding of virtual environments and, by extension, crowd simulation. In particular, this research will expand on the literature surrounding the use of genetic algorithms in the production of virtual environments and the use of crowd simulation to understand crowd movements within a hospital setting.

On a smaller scale, when generating crowd simulation environments, researchers typically try to either directly copy an existing environment or produce an environment that closely mimics such an environment. This research will take a different approach to the challenge of environment generation. Rather than take one of the more common approaches, I attempt to take a working crowd simulation designed for researching the movement of doctors, nurses, and patients in an emergency room setting and fit it to different computer generated environments. The goal here is to generate an emergency room floor plan that is "superior," or more conducive to the necessary movement of doctors, nurses, and patients, to floor plans existing in the real world.

In order to achieve the goals of this research, I have implemented a genetic algorithm and a procedural generation algorithm to create and optimize floor plans for the operation of our crowd simulation. The genetic algorithm, as described in the theory section, serves to evolve and optimize the floor plans that are generated by the procedural generation algorithm (also described in the theory section.) By the combination of crowd simulation, a genetic algorithm, and a procedural generation algorithm, we should be able to evolve more efficient floor plans for hospital emergency rooms.

II. Background

Crowd simulation, as a field of research, is vital to understanding the movement of people through an environment, virtual or physical. João Almeida, Rosaldo Rosseti, and António Coelho make a great case for crowd simulation research, discussing applications of this research, including creating realism in video games and similar entertainment mediums, analysing casual crowd movement, and developing greater understanding of groups in emergencies (Almeida et al., 2013). They write that crowds under stress move faster, with individuals disregarding concerns such as keeping distance from one another, resulting in a different behavior for the crowd as individuals search for direct paths. As the research of my project involves individuals in emergency situations, the environment of the crowd simulation and the ability of individuals to find direct paths is crucial.

In the area of virtual environment generation, the body of knowledge is vast. Ricardo Lopes et al., acknowledge that one of the main difficulties of producing floor plans is in placing and sizing the rooms (Lopes et al., 2010). They chose to resolve this issue with an algorithm that places and sizes rooms on a grid, with a set building facade. This is not the only solution, and many other researchers have developed other methods of generating a floor plan. For example, Rodrigues et al. use an L-system in combination with architectural legal rules to produce believable house layouts (Rodrigues et al., 2008).

Perhaps more relevant to this project are the researchers whose efforts focus on the coupling of virtual environments and evolutionary algorithms. Ruizhen Hu et al. report that AI and machine learning are finding integration with computer generated environments (Ruizhen et al., 2020). Their research uses a deep neural network called Graph2Plan and initial user input to generate house floor plans. Other researchers have tried different machine learning techniques in their floor plan generating processes. For example, Paul Merrell, Eric Schkufza, and Vladlen Koltun have taken advantage of trained Bayesian networks to automatically create full buildings, complete with fleshed out interiors (Merrell et al., 2010). Thus, AI and machine learning techniques are known to be applicable in creating realistic buildings in virtual environments.

Crowd simulations, virtual environment generation, and evolutionary algorithms each have a history of their own, however these three research topics have not been used all together in the manner that this research seeks to combine them, in the area of emergency medicine. Thus, the combination thereof has the potential to bring new knowledge and opportunity to the research community.

III. Theory

Genetic Algorithms, Procedural Floor Plans, and Crowd Simulation

Given the historical pedigree of this area of research, this project seeks to take the existing background in floor plan generation and extend its uses and boundaries. The idea behind this research is to evaluate whether the crossroad between genetic algorithms and crowd simulation is a fruitful area of study. Before we delve into what this may look like, it may be relevant to discuss the theoretical foundations of both genetic algorithms and crowd simulations.

A genetic algorithm makes use of some of the principles of natural selection and evolution to find an optimal solution to a given problem. Such algorithms continually create “generations” of new subjects - floor plans in our case - each of which is evaluated according to how effective they are at solving the problem. The better the rating, the more likely that particular subject will pass on its traits in the next generation; this kind of selection, as it is used in the genetic algorithm in this research, is known as “Elitism” (Saini, 2017). Other aspects of evolution, including mutation and chromosomal “crossing over” are modeled as well. The result is an algorithm that can explore the breadth and depth of possible solutions to a problem and produce an optimal solution.

Genetic algorithms have found limited use in floor plan generation before. The most relevant applications of genetic algorithms to floor plan generation are research articles such as that presented by Hamide Dalgic et al., wherein genetic algorithms were used to optimize the layout of supermarket shelves in relation to known customer shopping behaviors (Dalgic et al, 2017). The aforementioned genetic algorithm was used to block out the locations of shelves in a grid, representative of the supermarket floor. Notably, this genetic algorithm produces a floor plan that is a variation of a set space rather than a production of an entirely new floor plan.

In a different realm of theory, genetic algorithms rely heavily on a concept called "procedural generation" to operate. Procedural generation is a broad concept where an algorithm is made for the purpose of creating something useful without much human involvement. The application of procedural generation is powerful and varied, finding use across many disciplines (Ramsey, 2019). Genetic algorithms rely on procedural generation for the automatic and controlled creation of each subsequent generation of subjects. In the context of this project, procedural generation is used to generate various floor plans based on only a few inputs.

Procedural generation is a concept that can stand separated from genetic algorithms, and in the realm of floor plan generation, it often does. In Johan Melin and Daniel Bengtsson's comprehensive thesis on floor plan generation for game environments, they delve into great depth on the various methods of procedural floor plan generation (Melin and Bengtsson, 2016). They define five major procedural generation methods - subdivision, inside-out, growth, tile placement, and dense packing. Each method has merits and drawbacks, though the method relevant to this project is subdivision.

Subdivision, in its simplest form, is the creation of subsections from a building's facade (or outer walls). From those subsections, further subsections can be made. This process repeats as necessary until the remaining subsections can be considered to be the rooms, halls, and open spaces of a building.

The procedural floor plan generation method utilized in this project follows the subdivision method. The facade is divided into major subdivisions (see fig. 2), which are then divided into individual rooms. This has major implications for the functionality of our genetic algorithm. While the genetic algorithm can produce parameters that will allow for extensive variance from one floor plan to the next, the general shape and structure of the floor plan will always look similar to the model in Figure 2. The facade may be a different size, and the subdivisions may be larger or smaller, with more or fewer rooms, but the subdivisions and hallways will always be in that arrangement.

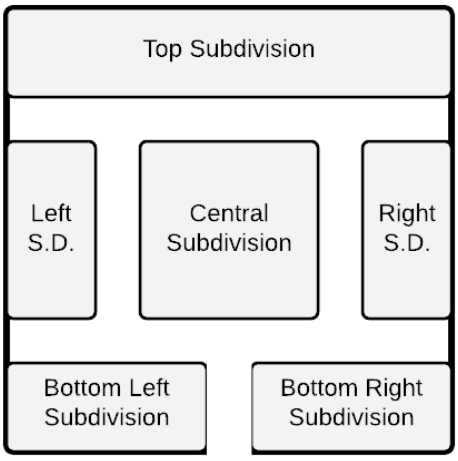


Fig 2. A diagram showing the major subdivisions in the emergency room floor plan generator.

This limitation means that the search space of the genetic algorithm, or the variety of results possible with the given parameters, is restricted to the common shape of most modern hospital emergency rooms. This may limit the innovative extents of the genetic algorithm. However, this does not mean that the genetic algorithm is without worth. Within the search space provided by the procedural floor plan generation algorithm, we may still find more optimal arrangements of rooms, uses of space, and movement paths for the crowd simulation.

The final major component of this research is the crowd simulation itself. Crowd simulation is a field of research into the movement of people in different environments and implementing these behaviors in a simulation. Through simulation, the behavior can be studied and tweaked with changes in the environment or circumstances. In the lab of Dr. Brian Ricks, crowd simulation is our primary area of research, which is reflected in our hospital emergency room simulation.

Heuristics and Fitness Functions

This research began with the genetic algorithm decoupled from the crowd simulation. The purpose of the genetic algorithm was to produce a single floor plan that had been optimized to cut down on the time that simulation agents spent travelling. Being decoupled from the crowd simulation meant that heuristics had to be developed to estimate what might make the crowd simulation more efficient. As such, values including number of rooms, hall width, and building size were manipulated to produce more fit floor plans. These heuristics were not accurate enough for our purposes, and we revised them.

Our next heuristic was to actually run the in-development crowd simulation to completion on each generated floor plan. Floor plans that finished faster were deemed more fit in the genetic algorithm. The crowd simulation, at this point, was unable to fully mimic the complex behaviors of hospital patients, nurses, and doctors, so a stand in simulation was implemented. In this simulation, an agent would be spawned in each room and their goal was to leave the hospital as quickly as possible. When all agents had left the hospital, the simulation would end.

Though this clearly is not a hospital emergency room simulation, it is useful in some ways. Notably, such a simulation has many agents moving through halls at the same time. The genetic algorithm compensates for this by widening the halls and doors, allowing for agents to get around each other without bumping into one another. Perhaps an unintended side effect of this simulation is that, since an agent spawns in each room, the genetic algorithm seems to try and minimize the number of rooms, and therefore the number of agents trying to escape. We were in need of a more accurate simulation, so the crowd simulation became a priority to get in working condition.

After working to get the simulation operating correctly with the genetic algorithm, we finally had an accurate fitness function to test the success of each layout. This fitness function is critically important to achieving the goals of this research, as it acts as the genetic algorithm's "natural selection" process. Layouts that are not as efficient, as determined by the fitness function, are less likely to pass their information on to the next generation. It is important to note that the simulation, at the time of this writing, only supports treating one patient at a time. This must be kept in mind as it may affect the results of the genetic algorithm.

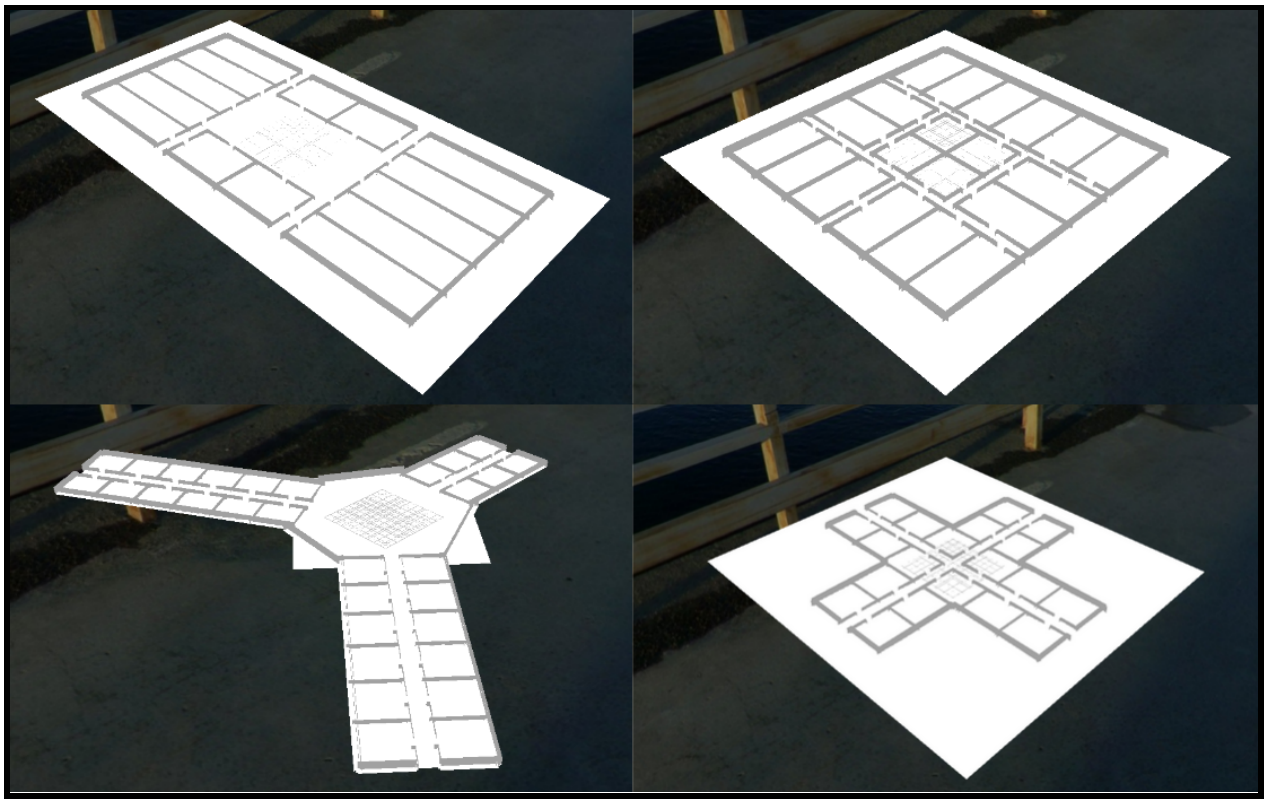
Ultimately, the combination of procedural generation, a genetic algorithm, and the working crowd simulation should be enough to produce meaningful results in the form of hospital layouts that are more efficient. The theory discussed in this section provides a basis for the implementation and results of the research of this project.

IV. Implementation

In the research process, two of the three working components were developed from the ground up. The procedural generation and genetic algorithms were written almost entirely from scratch. The crowd simulation, on the other hand, is and has been a work in progress by Dr. Ricks and students in his lab since well before I joined. This crowd simulation is complex and beyond my ability to describe effectively. Nevertheless, it is an important tool in this research and should another researcher wish to replicate this project, a similar crowd simulation will be needed.

Procedural Floor Plan Generation Algorithm

This procedural generation portion of this project began small, with a javascript file whose purpose was to write the details of a flat plane to an OBJ file. This process is rather involved for something so seemingly simple. OBJ files read like a different language with different letters and letter pairings specifying what, precisely, will appear on screen when the OBJ file is viewed. The numeric values that follow can describe vertex location, vertex pairs, object faces, and more.



Using what I had learned about OBJ files, I created procedural generation algorithms that could create a few different kinds of floor plans (fig. 3). The methods of floor plan generation employed in these algorithms is known as subdivision, which is discussed in detail in the theory section. This particular floor plan generation method was appealing for this project for a few reasons. The first of these is that hospitals are generally quite expensive to build, which means that we must generally keep an existing building facade (or outer wall.) Another reason is that many of the other existing methods of floor plan generation, such as growth, tile placement, and dense packing, may result in floor plans that are unsuitable for the organization of hospitals, which have a very particular structure, in comparison to the variance of a house floor plan or other similar floor plans.

Fig 3. The four broad types of floor plan generated by the procedural algorithms of our project.
Top Left: "H-Layout," named for the shape of the hallways.
Top Right: "Inner Circle Layout," which includes halls surrounding a central pod of rooms.
Bottom Left: "Y-Layout," with three branches off a central area.
Bottom Right: "X-Layout," with four branches off a central point.

While the subdivision based procedural generation method is outlined briefly in the theory section, the actual implementation needs further explanation. With a facade defined by the building width and building length parameters, the facade is represented within the algorithm as a "space," which is essentially two points marking the top left and bottom right of the area. From here, the facade's space is split into six smaller spaces and the hall spaces between them, as shown in Figure 2. The spaces that the halls fill are not marked as empty space, whereas the six other spaces are stored together for later use. Each empty space then is sent through a different method based on how many halls it is surrounded by. These methods fill the given empty space by subdividing it further into smaller spaces according to the halls that surround it. This step is necessary in my implementation of the algorithm because the rooms and their doors are generated in the OBJ file in the next step. Once the spaces are subdivided with the knowledge of which sides are adjacent to halls, a room is created.

This particular implementation of subdivision floor plan generation, while technically effective, is limited in the sense that it is not easily changed to produce profoundly different results. Essentially, the shape and locations of the major subdivisions split off from the facade will always be the same. This is a difficulty that I recognize limits the search space of the genetic algorithm to an extent.

After the procedural generation algorithm is complete, each of the rooms, as well as the hospital entrance, get a label. These labels are produced by iterating through an array of the filled spaces and writing a label to a JSON file. This process is difficult since there are a certain number of required labels that are necessary for the correct operation of the crowd simulation, however the number of rooms produced by the genetic algorithm's call to the procedural generation algorithm is impossible to know in advance. Since we need to maintain a measure of stability between generations while simultaneously optimizing the placement of these labels, I have made use of perlin noise. Perlin noise is a function with outputs that seem pseudorandom but are actually stochastic, meaning that a given input will always produce the same output. Furthermore, similar inputs will give outputs that are only slightly different. In this manner, I used perlin noise in conjunction with a "label value" parameter in the genetic algorithm to allow for the controlled assignment of required labels such that the genetic algorithm could have a measure of control over which locations in the hospital became which rooms.

Genetic Algorithm

The genetic algorithm, which makes use of the procedural floor plan algorithm, was similarly developed by myself in Dr. Ricks' lab entirely for use with our crowd simulation research. In the following paragraphs, the implementation and design choices of this genetic algorithm will be described in detail for reproduction by other researchers and interested parties.

Value Name	Minimum	Maximum
Hall Width	3	13
Door Size	3	8
Building Width	100	200
Building Length	100	200
Mid-Ratio	0.3	0.6
Max Room Size	10	20
Label Value	0	1

Fig 4. A chart displaying the value names, as well as the minimum and maximum range assigned for the trial described in the results section.

The genetic algorithm is a program that takes in three main parameters that change its functionality. The first parameter is the search space of the genetic algorithm. This search space is an array of seven labeled values with a minimum and maximum bound for each. The purpose of these values is to define the range of floor plans that can be generated. For example, when a floor plan is generated by the genetic algorithm, it may decide on any number between 3 units and 13 units for the width of the hall.

While most of these values are self explanatory, the last three in Figure 4 may need further explanation. The Mid-Ratio is a percentage that determines the amount of space that the central subdivision (see fig 2) is assigned to take up. Max Room Size is a value used when an area needs to be subdivided into rooms. Each room in the subdivision will be the same size and the algorithm will divide the space into the

minimum number of rooms such that their hall-facing sides do not exceed the Max Room Size. Lastly, the Label Value is used to control the perlin noise, as described in the procedural floor plan generation section.

The next parameter used by the genetic algorithm is the population size, which determines the number of floor plans that are generated and tested in each generation. In the research article, "Influence of the Population Size on the Genetic Algorithm Performance in Case of Cultivation Process Modelling," by Olympia Roeva, Stefka Fidanova, and Marcin Paprzycki, they attempt to find the optimal population size for genetic algorithms (Roeva et. al., 2013). They note that the best population size will depend on the problem being optimized, the specific version of the problem, and the time required to solve the problem. In their problem, they find that a population size of 100 is a good compromise between efficiency and accuracy in their algorithm. The circumstances are considerably different for the genetic algorithm in my research. The most impactful difference is the computational time spent running the crowd simulation on each of the floor plans produced by the genetic algorithm. This is not a quick process, and with a population size of 20 floor plans, evaluation of fitness could take upwards of five minutes. Increasing the population size would cause a similar increase in the runtime, making 20 floor plans a compromise in the thoroughness of the genetic algorithm in favor of a less computationally expensive algorithm.

Lastly, the number of generations establishes the number of times that the genetic algorithm loops before deciding on an optimal floor plan. This has an impact similar to population size on run time and results, although the number of generations acts as more of a cutoff point on the algorithm rather than an expansion of genetic diversity within the algorithm. For the purpose of this research, I chose to run the algorithm for 100 generations, as any more would be prohibitively time consuming.

With the parameters of the genetic algorithm defined, we can now explore the genetic algorithm loop. The genetic algorithm in this project can be broken down into five major processes, including one initializing process, as seen in Figure 5. Each process will be described to build a complete understanding of the algorithm.

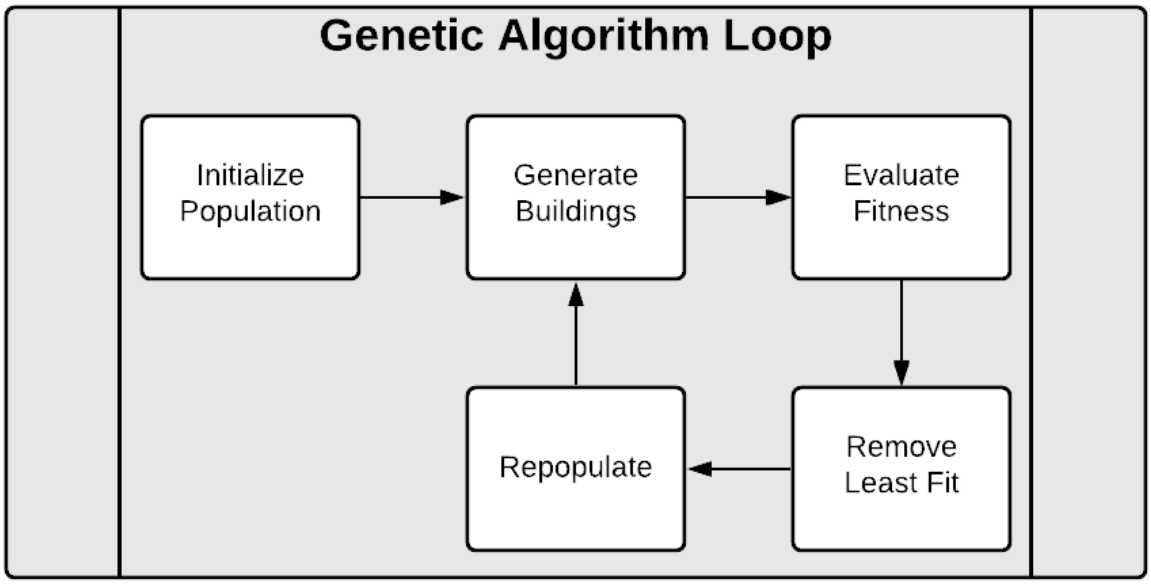


Fig 5. A depiction of the basic loop that the genetic algorithm executes.

The process of initializing the population is, perhaps, the simplest part of the algorithm. In this beginning step, each member of the population is generated by producing seven random numbers between zero and one and storing these numbers together in an array. Each of these numbers corresponds to a value in the search space. Once this process is complete, the algorithm moves on to actually generating the buildings.

The buildings are each generated by passing the values produced in the previous step, adjusted to the range given by the maximum and minimum associated with the value, as parameters to the procedural floor plan generation algorithm. The result is an OBJ file and the supporting JSON file with the information necessary to run the crowd simulation on the newly generated building. Each building is then assigned a worker thread and each layout is simultaneously tested to determine how long it takes for the simulation to complete. If a simulation takes more than 15,000 ticks of time, it is terminated and reported as having taken 15,000 ticks. This significantly reduces the run time, as sometimes the crowd simulation can take prohibitively long to resolve. When all of the simulations are complete and have reported their number of ticks, the algorithm moves on to evaluate the returned fitness values.

In the fitness evaluation phase, the most fit layout is determined by searching for the lowest time reported. Then, data from this generation is recorded for later analysis. Each floor plan is then assigned a fitness value, which is determined by dividing the best result by the result of the given floor plan. This will result in the best floor plan having a fitness value of ‘1’ and other floor plans having a value less than one.

From here, the fitness values are used to remove the least fit of the population this generation. In this algorithm, the five most fit floor plans are preserved for use in creating the next generation. In the final repopulation phase, the population is developed as follows:

- The best floor plan of the previous generation is added to the new population.
- Each of the five saved floor plans is mutated by randomizing one of the search space values until it passes a test of uniqueness (described later).
- The five saved floor plans are also matched with each of the other saved floor plans to perform the crossover operation. In this operation, for each search space value, one of the paired floor plans contributes their search space value. If the resulting floor plan does not pass the test of uniqueness, it is subjected to the mutation process until it passes the test of uniqueness.
- The remainder of the population is filled by randomly generated floor plans until the population meets the given population size.

The test of uniqueness takes in a floor plan's search space values and compares these values to the other members of the population. For each comparison, the difference between each search space value is calculated, then the total difference is divided by the number of search space values (seven, in this case.) If this value is greater than $0.1 * e^{-0.1n}$, where n is the current generation number, then the proposed search space value is sufficiently different. If the value passes this test for each other member of the population, then the test of uniqueness is passed.

The equation $0.1 * e^{-0.1n}$ functions as a key component of the genetic algorithm. It serves as an answer to the issue of exploration versus exploitation in evolutionary algorithms. Matej Črepinšek, Shih-hsi Liu, and Marjan Mernik touch on the idea of achieving a good balance of exploration and exploitation in such algorithms, emphasizing the importance of this balance to the success of an evolutionary algorithm (Črepinšek et al., 2013). In this genetic algorithm, I opted to take a simple approach. The equation given ensures that, in the early generations of the algorithm, no two floor plans will turn out to be within a certain degree of similarity. This serves to promote exploration within the genetic algorithm, as individuals within the population have to be distinct from each other. As the algorithm progresses, the equation returns a smaller value, allowing for more similar floor plans. This acts as an exploitation process within the genetic algorithm. Essentially, the algorithm explores the search space as a whole, and then it focuses on the area(s) that show the most promise.

The genetic algorithm loop is completed by returning to the "generate buildings" phase, if necessary. In combination with the procedural generation algorithms previously described, the genetic algorithm can produce interesting and optimal results.

V. Results

The results described in this section are the outcome of a 100 generation, 20 population size run of the genetic algorithm. The algorithm was provided with the "Inner Circle Layout" version of the procedural floor plan generator, shown in Figures 2 and 3. The crowd simulation, as mentioned previously, takes only one patient through the hospital emergency room simulation, where they are assisted by medical staff. Figure 6 shows an image of the most optimal floor plan at the end of 100 generations, which took approximately eight hours to complete.

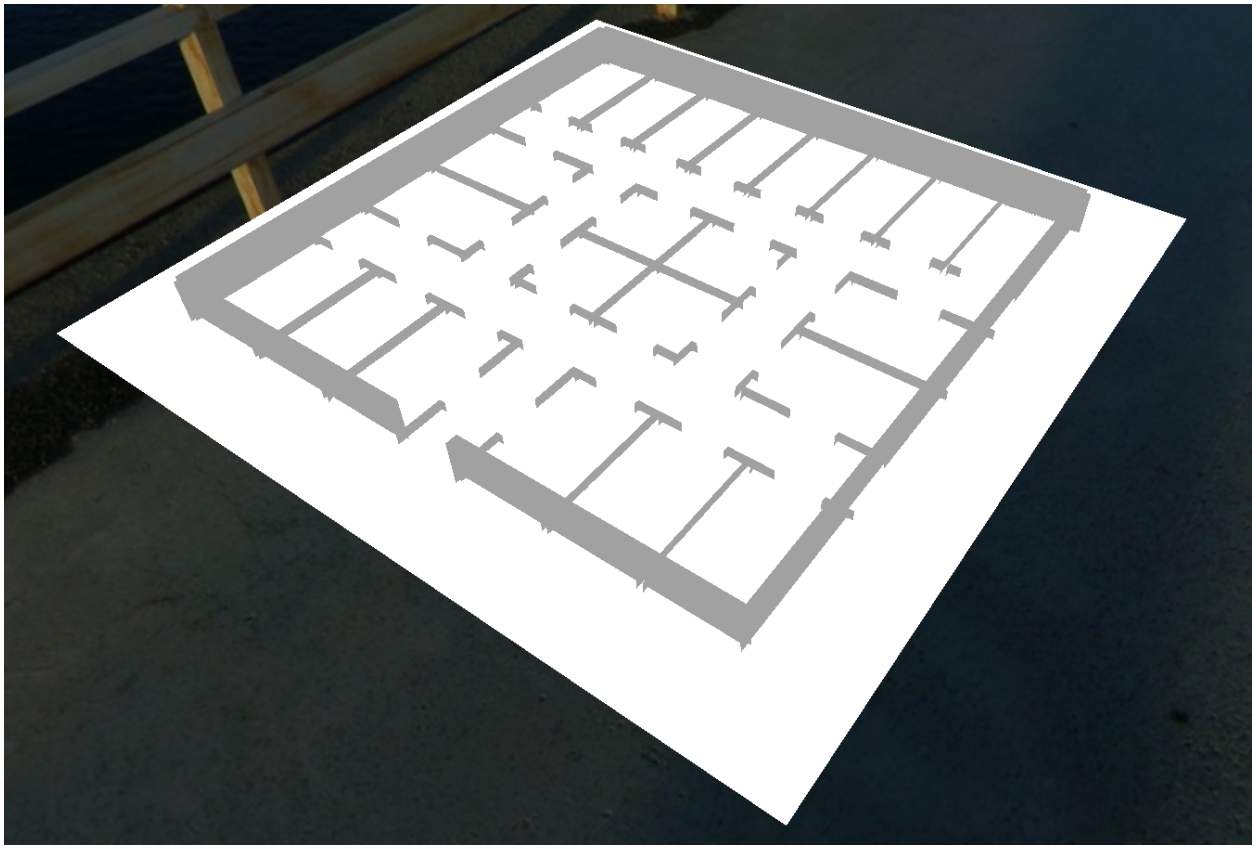


Fig 6. The most optimal floor plan after 100 generations at a population size of 20. The simulation completed in 6670 ticks for this floor plan.

Value Name	Range	Most Optimal Value
Hall Width	3 - 13	11.20
Door Size	3 - 8	7.96
Building Width	100 - 200	105.13
Building Length	100 - 200	103.06
Mid-Ratio	0.3 - 0.6	0.32
Max Room Size	10 - 20	12.87
Label Value	0 - 1	0.97

Fig 7. A chart displaying the value names, range, and the values of the most optimal floor plan after 100 generations at a population size of 20.

The resulting most optimal floor plan's values are displayed in Figure 7. It is important to bear in mind that each of these values are tied together, meaning that as a group these values are optimal. If a single one of those values were changed, it will affect how optimal the values are as a group. With that in mind, we may be able to draw some conclusions about the simulation and algorithms from these values.

Of note are the values of Hall Width, Door Size, and Building Width/Length. These values convey that the algorithm seemed to try and reduce agent path length by minimizing the building size and trending toward a fairly open floor plan.

Another interesting value is the Mid-Ratio. Recall, the Mid-Ratio determines how much space the middle section takes up in the floor plan. The genetic algorithm seemed to minimize the Mid-Ratio. This fact, in conjunction with the large Hall Width value, may suggest that a larger Mid-Ratio did not leave enough room for both the large Hall Width and the outer rooms. Rather than compensating for a larger Mid-Ratio with a reduced Hall Width, the genetic algorithm optimized for larger halls.

The Max Room Size value is interesting in that, out of all the values, it is the only one that didn't heavily favor one side of its range. It still favored the lower end of the range, however, it wasn't minimized despite having plenty of generations to change. Since Max Room Size is important in determining the number of rooms, these results would suggest that a Max Room Size near 13 units, which would produce a larger number of rooms than the midpoint of 15 units, balances having enough rooms for medical staff to have their necessary room close by while avoiding having so many rooms that it causes issues with the procedural floor plan generator.

The following section will shift focus from the resulting most optimized floor plan to the data and comments surrounding optimal floor plans for each generation. This data was obtained by outputting and compiling information about the most optimal floor plan in each generation.

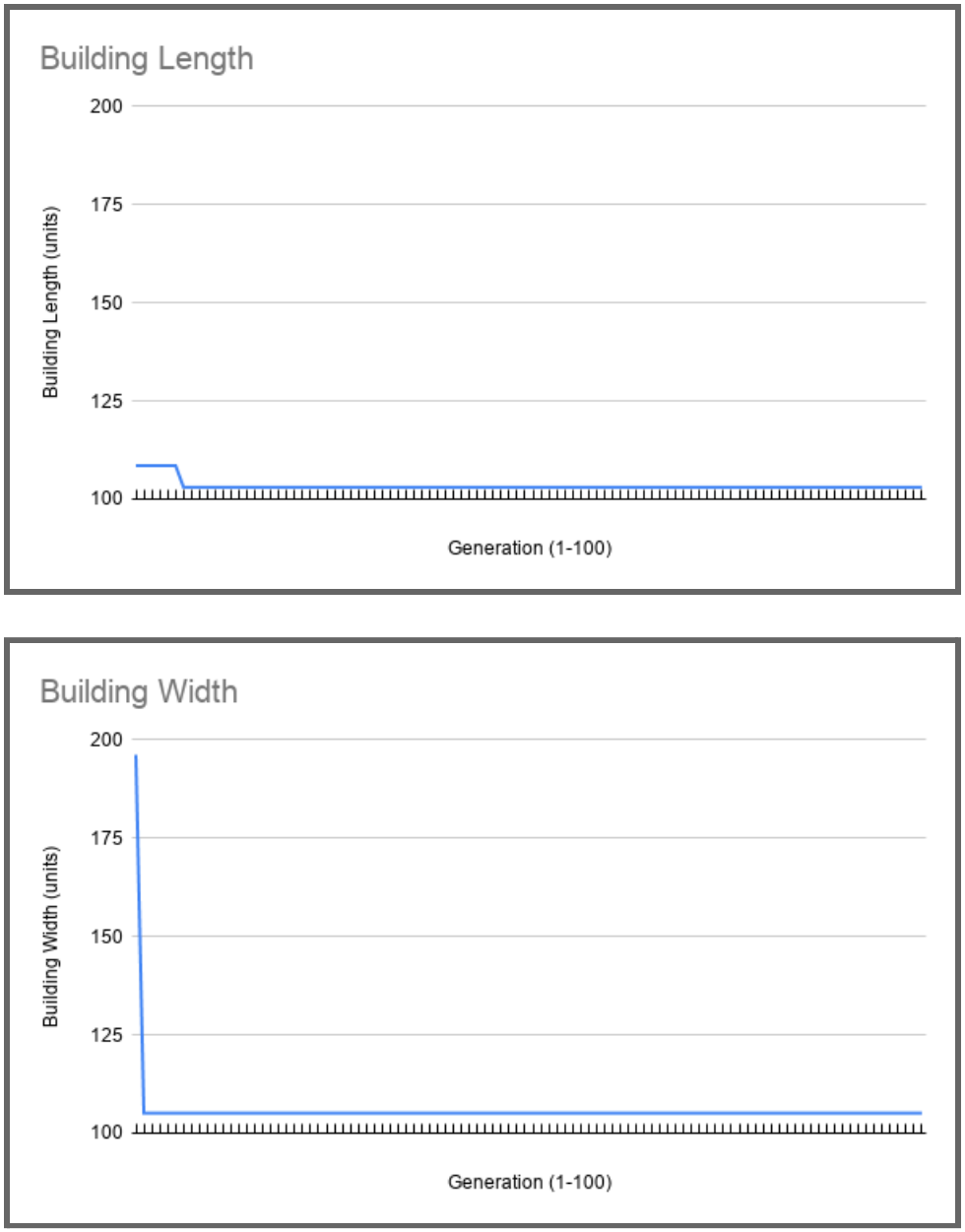


Fig 8 & 9. The Building Length and Building Width of the optimal floor plan over the course of 100 generations.

As can be seen, the genetic algorithm very quickly minimized both the Building Length and the Building Width. These values remained minimal throughout the generations. This shows that the genetic algorithm heavily favors a smaller building.

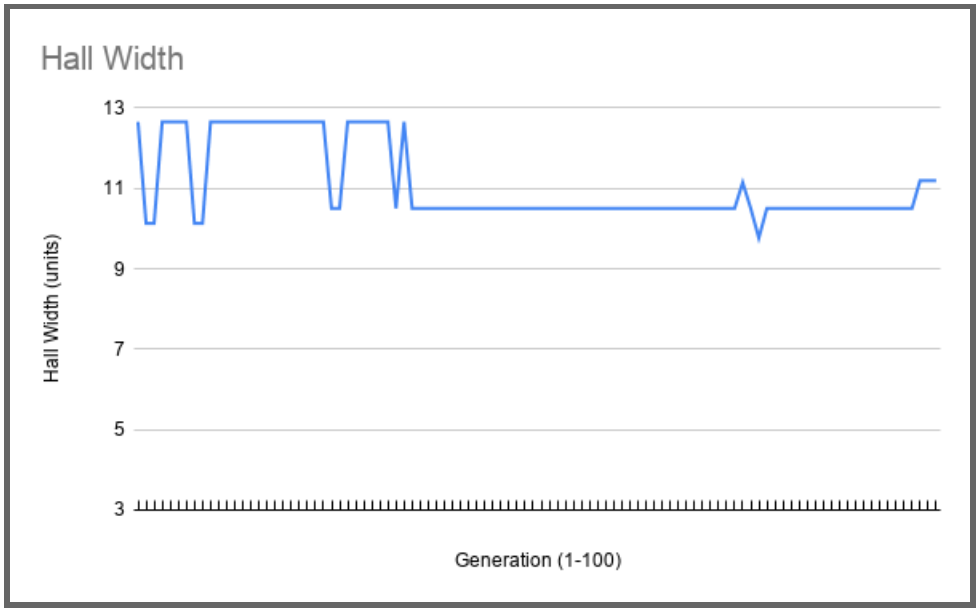


Fig 10. The Hall Width of the optimal floor plan over the course of 100 generations.

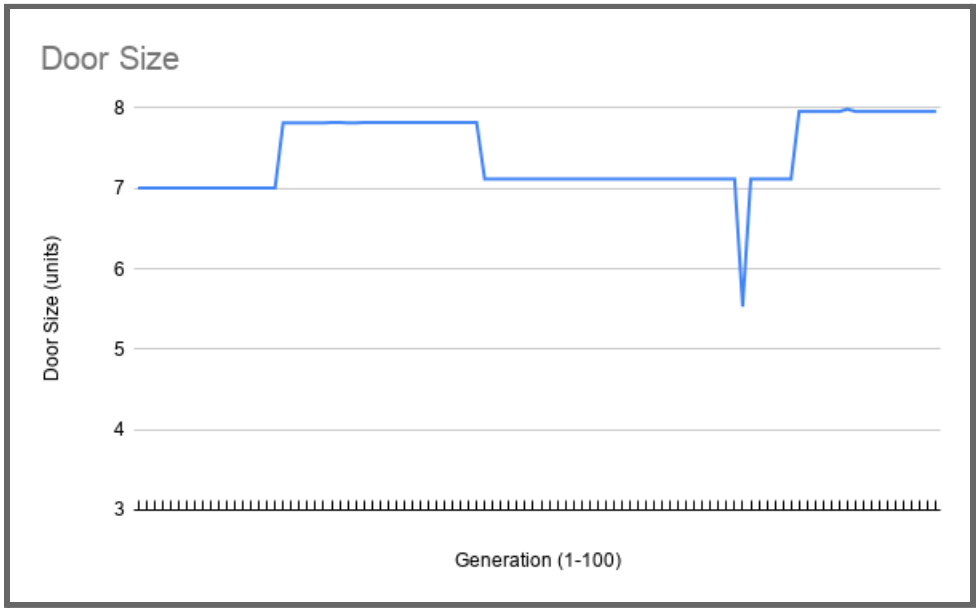


Fig 11. The Door Size of the optimal floor plan over the course of 100 generations.

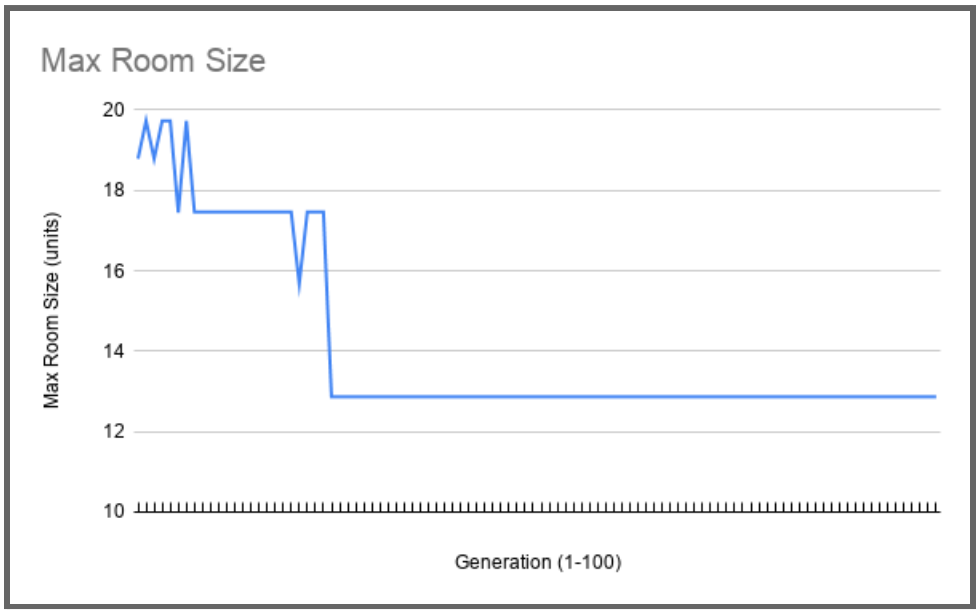


Fig 12. The Max Room Size of the optimal floor plan over the course of 100 generations.

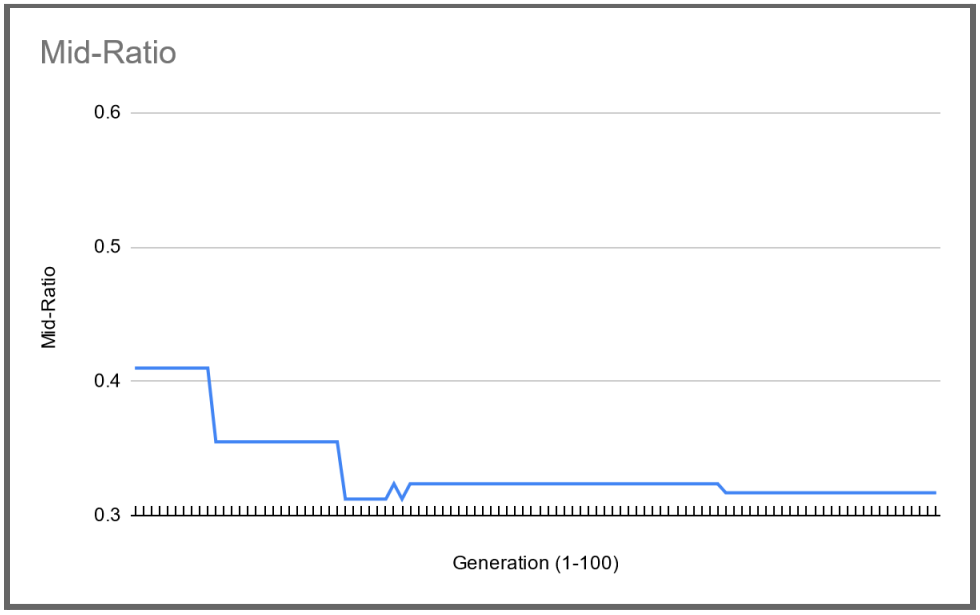


Fig 13. The Mid-Ratio of the optimal floor plan over the course of 100 generations.

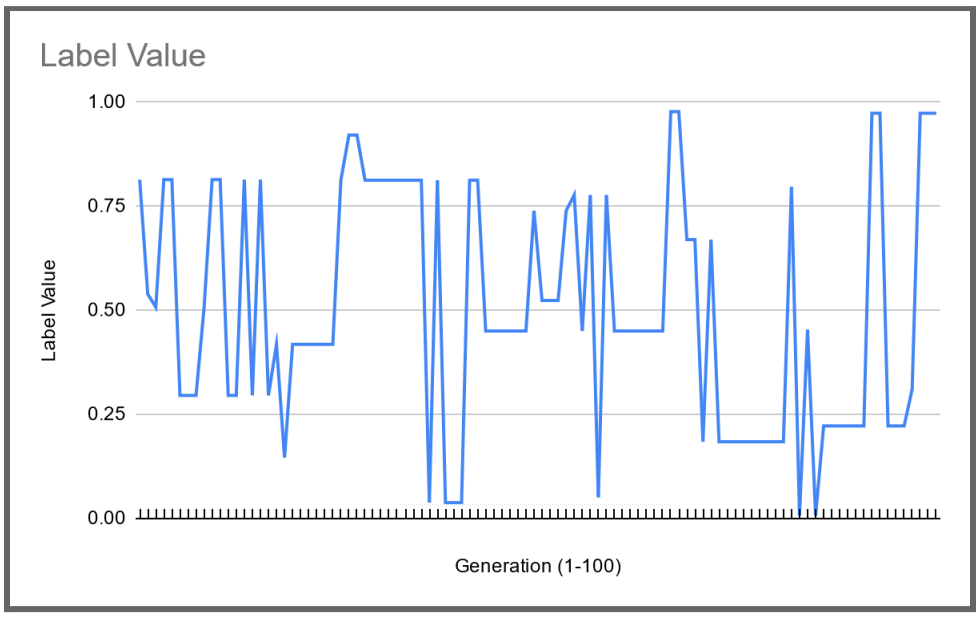


Fig 14. The Label Value of the optimal floor plan over the course of 100 generations.

The Label Value may need to be reimagined in similar research studies. While it was meant to bring stability to the room labels, it was not a stable value itself. This instability would cause labels to be less controlled and therefore more difficult to optimize. The difficulty in labeling the rooms comes from the variable number and positioning of the rooms. Since this is controlled indirectly by the other values passed to the procedural floor plan generator, this will likely be a difficult problem to solve moving forward.

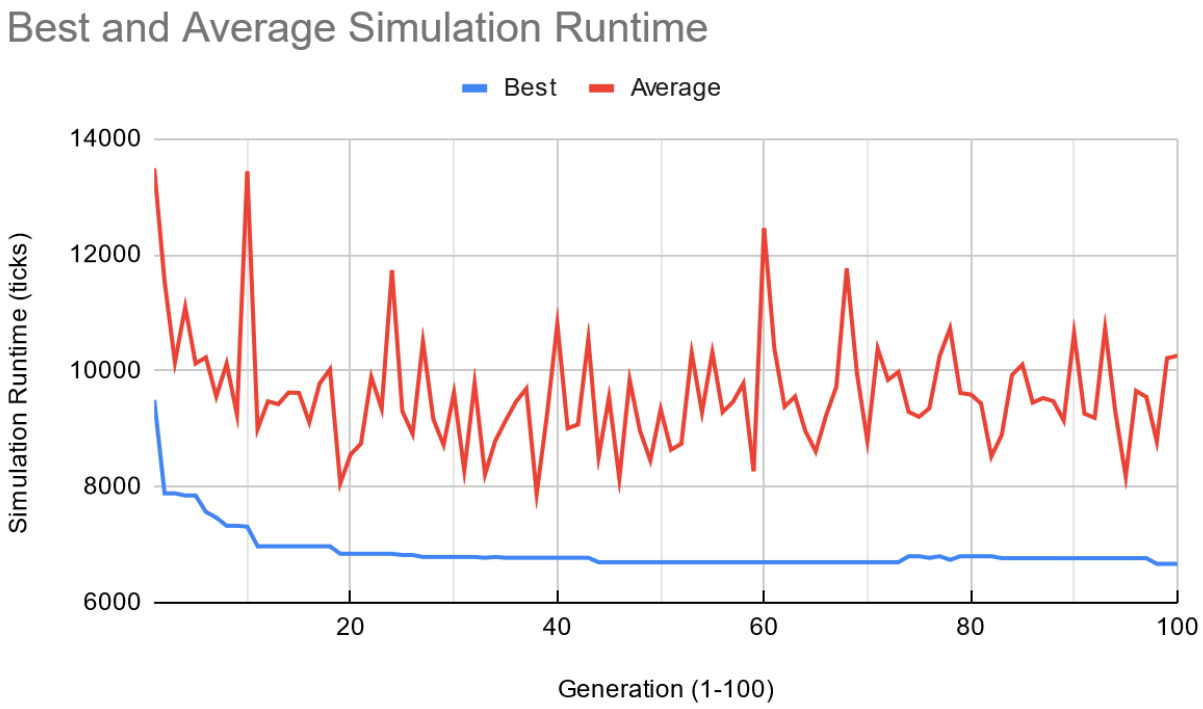


Fig 15. This graph shows the best and average simulation runtime over 100 generations.

This graph shows some of the more interesting of the data collected from the genetic algorithm trial. As can be seen, the genetic algorithm quickly develops a somewhat optimized floor plan as soon as generation 20. From there, the best simulation runtime trends downward. Notably, near generation 75, the best simulation runtime bumps up slightly. This is due to the chance that the previous best floor plan fails to resolve the crowd simulation in the same amount of time. A different floor plan outcompeted the previous best, resulting in the bump in the graph from around generation 75 to generation 97. Also worth noting is that, while difficult to tell, the best of generation 100 is still more optimal than the best of the range of generations around 60, with a simulation runtime of 6670 ticks and 6698 ticks, respectively.

VI. Future Work and Conclusion

In this report, I introduced a combination of a genetic algorithm, a procedural generation algorithm, and a crowd simulation in hopes of developing an environment that is more conducive to the effective flow of traffic through a hospital emergency room. From the results, we can tell that the genetic algorithm did, in fact, produce more optimal results. This is only the beginning, however, as the research presented here can and should be extended in a number of ways.

One suggestion that I would have for future research is that the results output by the genetic algorithm are compared to the same crowd simulation run on maps of existing environments. This would allow for researchers to draw conclusions about the potential effectiveness of produced floor plans in the real world.

Another suggestion is that one could fine tune a crowd simulation so that it runs faster. This would make trial runs of a genetic algorithm like ours much faster and allow for larger population sizes and numbers of generations. The results of such a work would make using a crowd simulation much more efficient as a fitness function in a genetic algorithm.

Also, as discussed previously, work could be done in making a more effective procedural generation algorithm or deciding on better parameters for the search space of the genetic algorithm. Either of these could result in a much more effective genetic algorithm for producing floor plans.

Modeling multi-story buildings is another direction that this research can be extended. Many of the buildings whose floor plans could be optimized with the heavy traffic of a crowd simulation are not a single floor, as was the case with the emergency room modeled in this research. Nor do emergency rooms exist in isolation. This being the case, the ability to work with multi-story buildings in this manner would prove a valuable extension of my research.

Acknowledgements

I would like to acknowledge my team members, Ryan Narducci and Sarath Kshatri for their diligent work on behavior trees and the web implementation, respectively, of our crowd simulation. Additionally, my boss and mentor, Dr. Ricks, has been a wealth of knowledge and encouragement in my academic pursuits. Lastly, I must acknowledge that this research endeavor was partially funded by the federal government, NSF # 1718139.

References

Almeida, João & Rossetti, Rosaldo & Coelho, António. (2013). Crowd Simulation Modeling Applied to Emergency and Evacuation Simulations using Multi-Agent Systems.

Črepinšek, Matej & Liu, Shih-hsi & Mernik, Marjan. (2013). Exploration and Exploitation in Evolutionary Algorithms: A Survey. ACM Computing Surveys. 45. Article 35. 10.1145/2480741.2480752.

Dalgic, Hamide & Bostanci, Gazi Erkan & Guzel, Mehmet. (2017). Genetic Algorithm Based Floor Planning System.

Lopes, Ricardo & Tutenel, Tim & Smelik, Ruben & de Kraker, Klaas Jan & Bidarra, Rafael. (2010). A constrained growth method for procedural floor plan generation.

Melin, J., & Bengtsson, D. (2016). *Constrained Procedural Floor Plan Generation for Game Environments* (Master's thesis).

Merrell, Paul & Schkufza, Eric & Koltun, Vladlen. (2010). Computer-Generated Residential Building Layouts. ACM Transactions on Graphics. 29. 10.1145/1866158.1866203.

Ramsey, Alexander. (2019). *Overview and Analysis of Procedural Content Generation in the Video Game Industry* [unpublished manuscript]. University of Nebraska Omaha.

Rodrigues, Nuno & Dionísio, M. & Gonçalves, Alexandrino & Magalhães, Luís & Moura, J. & Chalmers, Alan. (2008). Incorporating legal rules on procedural house generation. 59-66. 10.1145/1921264.1921279.

Roeva, Olympia & Fidanova, Stefka & Paprzycki, Marcin. (2013). Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. 2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013. 371-376.

Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Van Kaick, Hao Zhang, and Hui Huang. 2020. Graph2Plan: learning floorplan generation from layout graphs. *ACM Trans. Graph.* 39, 4, Article 118 (July 2020), 14 pages. DOI:https://doi.org/10.1145/3386569.3392391

Saini, N. (2017). Review of Selection Methods in Genetic Algorithms. *International Journal of Engineering and Computer Science*, *6*(12), 22261–22263. Retrieved from http://ijecs.in/index.php/ijecs/article/view/2562